RECEIVED
CENTRAL FAX CENTER
AUG 1 1 2008

U.S. Serial No. 10/659,457
Response to the office action of June 11, 2008

In the Specification:

Please amend paragraph [0007] in the following manner:

[0007] Mixed mode processors such as an ARM (previously, the Advanced RISC (Reduced Instruction Set Computer), and prior to that the Acorn RISC Machine, and now simply ARM) compliant processor, have two or more instruction sets such as, for example, a 16-bit instruction set (the Thumb instruction set) and a 32-bit instruction set (the ARM instruction set). Each of these instruction sets has advantages and disadvantages based on how the instruction sets are utilized. For example, the 16-bit Thumb instruction set typically encodes the functionality of the 32-bit ARM instruction in half the number of bits, thereby creating higher code density. An ARM instruction, however, typically has more semantic content than does a Thumb instruction. As is known to those having ordinary skills in the art, this means that an operation implemented with Thumb instructions may require more instructions to perform the equivalent operation implemented with ARM instructions (e.g., 40% more instructions). For example, to use a 16-bit immediate data location, the Thumb instruction set would require two more instructions to move the data into a register than would the ARM instruction set.

Please amend paragraph [0025] in the following manner:

[0025] The output of the language compiler 204 (i.e., the non-native instructions) is stored in the compiled binary 206 that may be, for example, Java byte code, .NET [[MSIL]] Microsoft Intermediate Language (MSIL), Perl byte code, etc. The compiled binary 206 comprises a plurality of non-native instructions that is in a format that the runtime environment may load, compile to native instructions, and then execute.

Please amend paragraph [0027] in the following manner:

[0027] The runtime environment 208 may be a Java virtual machine, a .NET common language runtime (CLR), a Perl virtual machine (e.g., Parrot), etc. The runtime environment 208 includes a [[JIT]] just-in-time (JIT) compiler 210 and an executing code 212. The compiled binary 206 may be loaded into the JIT compiler 210 to form a copy of the compiled binary 206 in a JIT memory location [[214]], which may be stored in the main memory device 108 in FIG. 1. Once the copy of the compiled binary 206 in the JIT memory location [[214]] has been loaded into main memory 108, the JIT compiler 210 may act upon the non-native instructions stored therein. For example, the non-native instructions may be acted upon by the JIT compiler 210 by generating the non-native instructions into native Thumb instructions by a Thumb code generator 216 or by generating the non-native instructions into native ARM instructions by an ARM code generator 218. The Thumb code generator 216 and the ARM code generator 218 may be components of the JIT compiler 210 that generate 16-bit instruction set code and 32-bit instruction set code respectively.

Please amend paragraph **[0030]** in the following manner:

[0030] In general, the example process 300 of FIG. 3 dynamically compiles instructions located on the main processing unit 102 of FIG. 1, although the instructions may have originated from the internet, [[POTS]] <u>the plain old telephone system (POTS)</u>, and/or other network(s) 118 of FIG. 1 or from the hard drive(s), CD(s), DVD(s), and/or other storage devices 116 of FIG. 1. More specifically, the example process 300 may operate in the JIT compiler 210 of FIG. 2. Preferably, the process 300 is embodied in one or more software programs which are stored in one or more memories and executed by one or more processors in a well known manner. However, some or all of the blocks of the process 300 may be performed manually. Although the process 300 is described with reference to the flowchart illustrated in FIG. 3, a person of ordinary skill in the art will readily appreciate that many other methods of performing the process 300 may be used. For example, the order of many of the blocks may be altered, the operation of one or more blocks may be changed, blocks may be combined, and/or blocks may be eliminated.